The Chameleon Project in Retrospective

Gerard J.M. Smit, Paul M. Heysters, Bert Molenkamp
University of Twente, department EEMCS, the Netherlands
g.j.m.smit@utwente.nl

Abstract

In this paper we describe in retrospective the main results of a four year project, called Chameleon. As part of this project we developed a coarse-grain reconfigurable core for DSP algorithms in wireless devices denoted Montium. After presenting the main achievements within this project we present the lessons learned from this project.

1. Introduction

In the Chameleon project we developed a framework for a tiled heterogeneous SoC for energy-efficient wireless devices such as handheld devices, mobile multimedia players, etc. In this framework a tile can be: a GPP/DSP, an FPGA, an ASIC or a coarse-grain reconfigurable core.

This project started in 2000 and ended this year. It initially involved two PhD students, a programmer and several staff members of the University of Twente. Later more projects related to this project were granted, so the research effort on energy-efficient architectures will not stop, but continues in other national and EU funded projects. We developed a coarse-grain reconfigurable core, called MONTIUM, for the DSP algorithm domain. This core can be used for typical DSP algorithms found in wireless handheld devices such as FIR/IIR filters, FFT algorithms, DCT algorithms, matrix multiplications etc.

In this paper we will look back at the results of this project. In section 2 the MONTIUM architecture is explained, in section 3 the results of the project are summarized, in section 4 we present the lessons learned form this project, and section 6 ends this paper with some conclusions.

2. Introduction to the MONTIUM

In this section a small introduction to the MONTIUM architecture is given, just enough to understand the remaining part of the paper, for more details we refer to [3].

In the Chameleon SoC organization conventional architectures are complemented by domain specific coarse-grain reconfigurable architectures. The MONTIUM architecture is devised as a vehicle to investigate the advantages and disadvantages of coarse-grain architectures. The key issue in the design of future ambient systems is to find a good balance between flexibility and high processing power on one side, and area and energy-efficiency of the implementation on the other side.

The MONTIUM targets the 16-bit digital signal processing (DSP) algorithm domain. A single MONTIUM Processing Tile is depicted in Figure 1. At first glance the MONTIUM architecture bears a resemblance to a VLIW processor. However, the control structure of the MONTIUM is very different. For (energy-) efficiency it is imperative to minimize the control overhead. This is for example accomplished by statically scheduling instructions as much as possible at compile time.

The lower part of Figure 1 shows the Communication and Configuration Unit (CCU) and the upper part shows the reconfigurable Tile Processor (TP). The CCU implements the interface for off-tile communication. The definition of the off-tile interface depends on the interconnect technology that is used in the SoC.

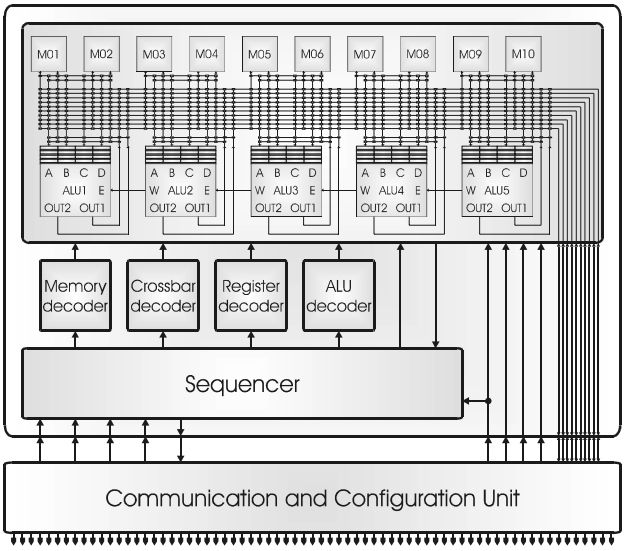

Figure 1*: The Montium Tile Processor.*

The TP is the computing part that can be configured to implement a particular algorithm. Figure 1 reveals that the hardware organization of the TP is very regular. The datapath of the ALUs has a width of 16-bits and the ALUs

support both signed integer and signed fixed-point arithmetic. The five identical ALUs (ALU1…ALU5) in a tile can exploit spatial concurrency to enhance performance. This parallelism demands a very high memory bandwidth, which is obtained by having 10 local memories (M01…M10) in parallel. A relatively simple sequencer controls the entire tile processor. The sequencer selects configurable tile instructions that are stored in the decoders (see Figure 1).

Each local SRAM is 16-bit wide and has a depth of 512 positions, which adds up to a storage capacity of 8Kbit per local memory. A reconfigurable Address Generation Unit (AGU) accompanies each memory. It is also possible to use the memory as a lookup table for complicated functions that cannot be calculated using an ALU, such as sine or division (with one constant). A memory can be used for both integer and fixed-point lookups.

A single ALU has four 16-bit inputs. Each input has a private input register file that can store up to four operands. The input register file cannot be bypassed, i.e. an operand is always read from an input register. Input registers can be written by various sources via a flexible interconnect. An ALU has two 16-bit outputs, which are connected to the interconnect. The ALU is entirely combinational and consequentially there are no pipeline registers within the ALU. Neighbouring ALUs can also communicate directly on level 2. The West-output of an ALU connects to the East-input of the ALU neighbouring on the left. The East-West connection does not introduce a delay or pipeline, as it is not registered.

3. Results of the MONTIUM project

In this section we summarize the main results of the project, for more details we refer to [3]. First the implementation results are given and after that the mapping of sample algorithms is given.

3.1. Implementation results of the MONTIUM

The ASIC synthesis of the MONTIUM TP was performed using the Philips CMOS12 process technology. This process has a (drawn) gate length of 0.13 µm and a density of 200 kgates/mm^2. For the local data memories and sequencer instruction memory of the MONTIUM TP embedded SRAMs are used. The embedded SRAM is an optimized component from a cell library.

For ASIC synthesis worst case military conditions are assumed. In particular, the supply voltage is 1.1 V and the temperature is 125°C.

Results obtained with the synthesis are:

- The area of the MONTIUM TP according to the synthesis tool is about 1.8 mm^2.
- With Philips' tools we estimated that the MONTIUM TP ASIC realization can implement an FIR filter at about 140 MHz or an FFT at about 100 MHz.
- The table below gives an overview of the average power consumption of a single MONTIUM tile. In the table there is a column for configuration power, power needed to load the input samples, power for the execution and power for the retrieving of the results.

Algorithm	Average power [µW/MHz]			
	Configure	Load	Execute	Retrieve
FFT64	386	229	541	208
FFT1024	386	228	577	203
FIR5	382	148	374	151
FIR20	389	150	420	152

Table 1: Average power consumption MONTIUM

3.2. Mapping algorithms of the domain

In first instance we mapped various algorithms to the MONTIUM (e.g. FIR, FFT, DCT, matrix multiplication etc.). To analyze the feasibility and flexibility of implementing multi-standard communication systems in the MONTIUM we have implemented key algorithms of the baseband processing for HiperLAN/2 (similar to IEEE 802.11a) [4], Bluetooth [5] and UMTS [8]. These three communication standards have been selected, because they are different enough to give an indication whether our approach is feasible or not and they are already part of ongoing research at the University of Twente.

We summarize the results here, more detailed information of the mappings can be found on [3]. The HiperLAN/2 receiver can be implemented in 3 MONTIUM tiles running on a fairly low frequency of less than 50MHz. In this way the baseband processing of one OFDM symbol is pipelined over multiple MONTIUM tiles, since in HiperLAN/2 every 4 µs a new OFDM symbol has to be processed.

The functionality of the Bluetooth receiver appeared to have a fairly simple signal processing part, which can be implemented in the MONTIUM quite well.

For a UMTS receiver we implemented a 4 finger Rake-receiver in one MONTIUM tile. For every four fingers extra we need an extra MONTIUM tile. In addition to that a tile is allocated for Turbo or Viterbi decoding.

4. Lessons learned in the Chameleon project

Looking back on the past 4 years of the Chameleon project we will address in this section the main lessons learned from this project.

4.1. Hardware and software co-design

Developing the hardware architecture of a coarse grain reconfigurable core like the MONTIUM is a complex but relatively straightforward process, however, the software

design flow / design methodology is not trivial at all. Looking back we strongly believe that hardware and software should be developed hand in hand because slight changes in the hardware architecture (e.g. adding a pipeline register) might have severe consequences for the software development flow.

In the first two years of the project we concentrated very much on the hardware architecture, and procrastinated the software design flow. After realizing this we put more effort on the software design trajectory. Additional funding was requested and, fortunately, granted for the software design flow. We believe this lesson could have been deadly for commercial reconfigurable companies, (see Chameleon systems).

We believe one of the reasons is that hardware architecture is already a well established field with reasonable mature development tools (VHDL simlators and synthesizers), but the development of 'compilers' for coarse-grain reconfigurable cores is not a main stream business. There are good compiler front-end tool-kits available, but the main problem is in the back-end of the compilers.

The back-end of our compiler is still work in progress in a related project called Gecko [3]. One of the lessons we learned in the mapping process is that it is important to start with regular algorithms: regular in the sense that there are many common operators (e.g. MAC operations), regular memory access patterns and communication patterns. Fortunately, DSP algorithms tend to be regular. We discovered that some standard compiler optimizations ruin the regularity of the original algorithm. Quite often these optimization algorithms have to be switched off. An example is a FIR filter, which is by nature quite regular; however, some compiler optimizations try to minimize the amount of additions by balancing the tree of adders. For a straightforward mapping of a FIR filter this optimization ruins the regularity of the MAC operations.

4.2. Energy overhead is everywhere

To make energy-efficient systems for wireless systems requires knowledge and coordination of several disciplines, ranging from computer architecture, compiler technology, algorithm domain: wireless baseband processing, protocols, multimedia algorithms etc.

Although we targeted at an efficient architecture for handheld devices, and thought we were energy-aware, we discovered that our initial design had some energy in-efficiencies. For example, during the first energy estimation we discovered that the clock consumed about 70% of the energy. For instance the configuration registers were clocked constantly, even while in the execution mode. An addition of a simple clock-gating circuit reduced the clock energy to less than 10% of the tile energy.

In the MONTIUM design locality of reference is used as one of the guiding principles to obtain energy-efficiency. The small local memories, for instance, are motivated by the locality of reference principle. The ALU input registers provide an even more local level of storage.

We found that there is a clear lack of good energy-estimation tools. For the FPGA design flow there are some power estimation tools available (e.g. Xilinx XPower), but for an ASIC trajectory this is not the case. For the energy estimation for the MONTIUM we had to rely on the kindly cooperation of Philips Research. They have good energy estimation tools (called Diesel), which is closely linked to their technology parameters. Accurate energy estimation tools in the design flow are required to develop energy-efficient systems. These tools are needed to predict the energy consequences of certain design decisions.

As we did not have these tools we were quite often forced to do coarse extrapolations for new algorithms using the results of the Diesel simulation of the FFTs and the FIR filters.

4.3. Know your algorithm domain

Building a domain specific coarse-grain core is an iterative process which means that you start of with an initial architecture based on your domain knowledge. With this architecture you start to map characteristic domain specific algorithms and you gradually improve this architecture. Building the algorithm domain knowledge is a very time consuming process, because even for the DSP algorithm domain this knowledge is sometimes quite distributed over various groups. For example, we discovered that people that develop algorithms for UMTS rake-receivers hardly communicate with people developing error correction algorithms e.g. Turbo decoding. There are ample possibilities for cross-discipline optimizations there, but it takes a rather long investment to obtain sufficient domain knowledge to be able to perform the cross-discipline optimizations [9].

4.4. How much flexibility do we need?

In a project like this it is difficult to find the right balance between flexibility and overhead. Too much flexibility leads to more chip area and more energy overhead and too few flexibility leads to mapping problems. Only after mapping typical algorithms of the algorithm domain you discover whether the design choices regarding flexibility were right. Below some examples are give that illustrate this flexibility struggle:

4.4.1 Bit-reversal for the FFT

Although we knew that bit-reversal is used in an FFT, we did not initially implement it, because we thought this would be too FFT specific and we thought we had the flexibility to do without. Later when performing the mapping of larger FFTs we discovered that we really needed bit-reversal.

4.4.2 Flexibility of level 1 of the ALU

Level 1 of the ALU initially had much more flexibility, but while mapping algorithms to the MONTIUM we discovered that we never needed this flexibility, and, moreover, it turned out that level 1 had a considerable energy and area overhead. So we reduced the flexibility of level 1.

4.4.3 Size of the decoders

The decoders as we have them right now are too big and the amount of configuration registers quite often limits the mapping of algorithms. This experience could only be obtained after having mapped several algorithms. In the next generation of the MONTIUM this will be corrected.

4.4.4 Register contents

In our ALU each input is connected to four input registers; in total there are 4x4x5=80 (16 bit) input registers. In the current design these registers cannot be filled at configuration time. After mapping several algorithms it turned out that it would be quite useful to fill at configuration time a register with a constant. In the current design these constants have to be loaded from memory, which requires memory space and more important configuration space. In the next version of the MONTIUM this will be improved.

4.4.5. Configuration space / adaptive systems

One of the features of dynamically reconfigurable systems is that they can change the configuration at run-time. In the herefore mentioned UMTS rake-receiver this is used to change at run-time the amount of fingers. However, this is only possible when the configuration space is small and the configuration can be changed incrementally. In the MONTIUM architecture the total configuration space is 2.6 Kbytes, and the configuration can be written as a static RAM memory. For a two-finger rake-receiver only 570 bytes of configuration memory needs to be written. Changing from two fingers to one finger only requires 40 bytes of reconfiguration. In this way an adaptive system can be realized, that changes its behavior at run-time.

5. Related work

Both academy and industry show interest in coarse-grained reconfigurable architectures. The Pleiades project at UC Berkeley [1] focuses on an architectural template for ultra low-power high-performance multimedia computing. In the Pleiades architecture template a general-purpose microprocessor is surrounded by a heterogeneous array of autonomous, special-purpose satellite processors. The Pleiades SoC design methodology assumes a (very) specific algorithm domain.

Quicksilver's adaptive computing machine (ACM) technology [7] is intended for low-power mobile devices. Their key observation (and we agree on that) is that algorithms are heterogeneous by nature. The architecture of the ACM comprises heterogeneous nodes of different granularities.

Silicon Hive [5] delivers reconfigurable accelerators that are customized for a certain algorithm domain. The reconfigurable accelerators are designed using a solid internal design flow that has evolved over a long time. Proprietary Silicon Hive tools automatically generate a C compiler for an accelerator.

Acknowledgements

We would like to thank Jos Huisken of Philips Research / Silicon Hive for facilitating the area and energy estimations. This research is supported by the PROGram for Research on Embedded Systems & Software (PROGRESS) of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the technology foundation STW.

6. Conclusion

In this paper we described in retrospective the main results of a four year project, called Chameleon, in which we developed the MONTIUM: a coarse grain energy-efficient reconfigurable core for DSP algorithms in wireless devices. We succeeded in fulfilling our main project objectives: small size (1.8 mm^2), flexibility (useful for a wide range of DSP algorithms), energy-efficiency (500 μW/MHz) and high performance (FFT butterfly in 10 ns). We have learned valuable lessons concerning the energy-efficiency, the hardware/software co-design, the algorithm domain and the flexibility.

7. References

[1] Abnous A.: "Low-Power Domain-Specific Processors for Digital Signal Processing", *Ph.D Dissertation*, University of California, Berkeley, USA, 2001.

[2] Baumgarte V., May F., et al.: "PACT XPP – A Self-Reconfigurable Data Processing Architecture", *Proceedings Engineering of Reconfigurable Systems and Algorithms*, pp. 64-70, Las Vegas, USA, June 2001.

[3] chameleon.ctit.utwente.nl

[4] ETSI, Broadband Radio Access Networks (BRAN); HiperLAN type 2; Physical (PHY) layer, ETSI TS 101 475 V1.2.2 (2001-02), 2001

[5] Bluetooth SIG, Specification of the Bluetooth System – Core, Technical Specification Version 1.1, 22 February 2001

[6] http://www.siliconhive.com

[7] http://www.qstech.com

[8] www.3gpp.org

[9] Smit L.T. "Energy-Efficient Wireless Communications" PhD University of Twente, ISBN 90-365-1986-1.